Streaming and Exploration of Dynamically Changing Dense 3D Reconstructions in Immersive Virtual Reality

Annette Mossel*, Manuel Kroeter†
Interactive Media Systems Group, Vienna University of Technology, Austria

ABSTRACT

We introduce a novel framework that enables large-scale dense 3D scene reconstruction, data streaming over the network and immersive exploration of the reconstructed environment using virtual reality. The system is operated by two remote entities, where one entity – for instance an autonomous aerial vehicle – captures and reconstructs the environment as well as transmits the data to another entity – such as human observer – that can immersivly explore the 3D scene, decoupled from the view of the capturing entity. The performance evaluation revealed the framework's capabilities to perform RGB-D data capturing, dense 3D reconstruction, streaming and dynamic scene updating in real time for indoor environments up to a size of $100m^2$, using either a state-of-the-art mobile computer or a workstation. Thereby, our work provides a foundation for enabling immersive exploration of remotely captured and incrementally reconstructed dense 3D scenes, which has not shown before and opens up new research aspects in future.

Index Terms: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities; I.4.8 [Computing Methodologies]: Image Processing and Computer Vision—Scene Analysis

1 INTRODUCTION

Capturing, mapping and 3D reconstruction of environmental geometry has been broadly studied in the past. With the emerge of low-cost commodity RGB-D depth imaging sensors combined with algorithms for capturing and mapping, dense 3D scene reconstruction became widely available. While there are a plethora of approaches for tracking and mapping of the environment, the *exploration* of dense surface reconstructions has not been subject of intensive investigation so far. This is particular true for 1) remote exploration of the reconstructed environment while capturing is still on-going, and 2) remote exploration of the reconstructed environment using immersive virtual reality (VR) input- and output devices.

Live and remote scene exploration can be beneficial in terms of costs, speed and safety. In combination with immersive VR technology, additional value is added in terms of natural viewing and improved spatial scene understanding [13]. These properties are vital to many real-world uses cases, such as rescue operation and remotely (guided) inspections. For rescue operations, for instance an autonomous aerial robot captures hazardous environments and transmits 3D spatial information to a human entity for exploration, such as first responder units outside the building. For inspections, one person can capture an environment with a handhold scanning device, while another person can remotely inspect the scene.

1.1 Motivation

State-of-the-art methods lack capabilities to enable reconstruction and live immersive scene exploration by two remote entities, where one is capturing and the other one is exploring the scene. Prior art for 3D scene reconstruction can be differentiated between 1) self-contained vs. distributed reconstruction, and 2) means for 3D scene exploration. Most prior art perform surface reconstruction with neither distribution nor build-in exploration functionality [18]. As a consequence, the resulting 3D model must loaded to an external viewing application for exploration, such as [16]. Alternatively, the capturing, mapping and exploration process is distributed and can be performed by two remote entities that are connected over a network. For data transmission, a trivial solution is to transmit the complete 3D reconstruction after the mapping process has finished. As a consequence, there is a temporal gap between capturing and exploration, which is in particular long for large scenes. Alternatively, the data can be incrementally streamed while the capturing process is on-going. Thereby, the exploring entity is enabled to immediately view the scene which allows for decision making and spatial knowledge acquisition from the very beginning. Therefore, the current view of the depth imaging device can be streamed while mapping and reconstruction are performed on the receivers's side [17]. As a consequence, the reconstructed scene can be explored, independent of the current camera's position. However, this principle is prone to network failures and bottlenecks. Once the network connection is lost, the reconstruction cannot continue and might fail due to camera tracking errors. Moreover, the capturing entity – i.e. a robot – cannot proceed since it does not get instructions from the server anymore, e.g. for navigation purposes. Alternatively, capturing, mapping and volumetric data transmission can be performed by the capturing entity, while the exploration side receives the data and reconstructs it so that both entities have the same 3D information. While this requires computational power on both sides, the capturing entity remains independent and can perform autonomous operations. In all the above mentioned approaches, 3D scene exploration is performed using traditional 2D in- and output devices, such as keyboard, mouse and screen which does not constitute a natural 3D user interface.

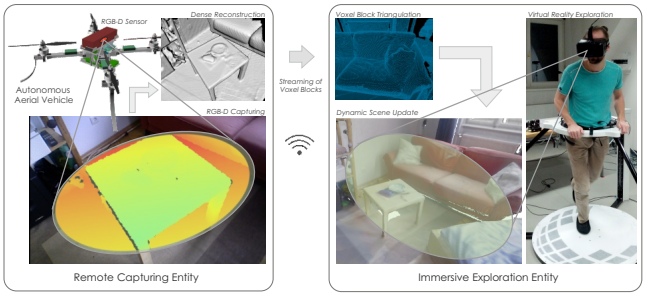

Figure 1: Our proposed framework for 3D surface reconstruction, streaming and immersive exploration by two remote entities.

1.2 Contribution

We position our work in the context of current research efforts to 1) reconstruct large 3D indoor environments using commodity RGB-D imaging hardware, 2) stream 3D surface models over the network

*e-mail:annette.mossel@tuwien.ac.at
†e-mail:manuel.kroeter@tuwien.ac.at

and visualize it, and 3) navigate through virtual worlds.

By interlocking research from these areas, we contribute with the proposed novel framework for 3D surface reconstruction, streaming and immersive exploration, as depicted in Figure 1. The framework partly builds upon prior art and integrates newly developed methods to enable the integration of two independent entities, 1) the remote capturing and mapping entity and 2) the exploration entity. Our framework provides simultaneous localization and mapping (SLAM) on the capturing side to allow for autonomous (robotic) operations. It transmits the volumetric data as voxel representation over the network and provides live exploration of an incrementally updating 3D environment by a human observer using VR technology. This work should inspire researchers and practitioners to further explore the opportunities of virtual reality technology to interact with dense 3D environments and consequently, to add additional value to 3D reconstructions and to apply it to novel use cases.

2 Related Work

Our proposed framework incorporates 3D surface reconstruction, streaming, extraction, visualization and immersive exploration. To the authors best knowledge no prior art exists that combines all aforementioned modules. However, each area itself has been well studied and the most relevant approaches are briefly presented in the following.

Dense 3D Surface Reconstruction 3D surface reconstruction methods that incrementally fuse depth imaging data into a single representation of the scene has been widely studied, especially since the emerge of Microsoft Kinect and the Kinect Fusion algorithm [10]. It estimates the camera pose with the Iterative Closest Point algorithm (ICP) where it uses a synthetic depth map from the current model to minimize camera drift. As a model representation, it uses a dense volumetric Signed Distance Function (SDF) that is stored in a regular 3D voxel grid on the GPU. To overcome the volume limitations of Kinect Fusion, Moving Volume approaches [12] were introduced that only stores the active volume – the current camera view – inside the GPU. The scene is still represented as a regular grid, but those parts of the scene that fall out of the current active volume are streamed to the CPU to free space for new data on the GPU. Voxel Block Hashing [11] only stores data close to the actual surfaces but without resorting to a scene hierarchy. It employs a spatial hash function which maps a 3D voxel position from world space to a hash table entry. Only data (SDF and color) for visible voxels is stored on the GPU memory, the other voxels are streamed to the CPU similar to the Moving Volume techniques.

Data Streaming The way how to compress and stream a reconstructed 3D environment heavily depends on the data representation and can either be lossless or lossy. Many related approaches send RGB-D data or images of the current reconstruction – as seen from the camera's viewpoint – over the network by employing standard image compression methods or algorithms specifically designed for depth data [2]. Alternatively, the acquired 3D point cloud can be transmitted, as shown by [6] that make use of spatial grids to compress the model. Streaming of volumetric reconstruction data stored as a SDF has not been in particular investigated by prior art. However, it is possible to apply compression methods targeted at arbitrary data. A volumetric SDF model can be effectively compressed using a lossless general purpose algorithm like run-length encoding or more advanced methods such as DEFLATE or bzip2.

Surface Extraction Upon data reception, the 3D reconstruction must be visualized to enable immersive exploration. As described in Section 3, a mesh representation is therefore desired that need to be extracted from the underlying model. Thus, the mesh extraction must be applicable for dynamic data and needs to be able to update in real-time upon model changes. Poisson Surface Reconstruction [8] is a popular approach to compute a mesh from a dense point cloud, however it lacks performance for surfaces that contain holes and noise for which the Greedy Projection Triangulation (GPT) performs more robustly. To reduce geometrical complexity of the underlying data structure, Whelan et al. [19] provides geometry simplification within an incrementally updated point cloud. To create a triangular mesh from a volumetric representation such as a SDF, the common approach is applying Marching Cubes [9]. A number of recent work focused on minimizing the amount of triangles produced by the Marching Cubes [14].

Visualization & Immersive Exploration Upon triangulation of the 3D reconstruction, the standard rendering pipeline using rasterization can be applied by using a state-of-the-art game engine, such as Unreal Engine or Unity 3D. Both incorporate advanced rendering techniques and physical effects, which are vital for exploration. While exploring a 3D scene using standard input and output devices (i.e. keyboard, monitor) only provide abstract input and 2D output, immersive virtual reality devices can be employed to facilitate a more natural user experience. Head Mounted Displays (HMDs) (i.e. Oculus Rift, upcoming HTC Vive) can be used to provide an immersive experience since the enable stereoscopic and egocentric scene viewing. Besides immersive viewing, navigation is key for exploration. Therefore, abstract input devices such as joysticks or game controllers can be employed. To provide a more natural navigational input, an omni-directional treadmill (ODT) (i.e. *Cyberith Virtualizer*, *Virtuix Omni*, *Infinadeck*) can be employed to improve matching between real and virtual movements as well as to provide proprioceptive feedback.

3 Methodology

The proposed immersive exploration framework comprises a server-client architecture and is depicted in Figure 2. It uses parts of the prior art framework InfiniTAM [7], extends it and adds and novel functionalities, as described in the following.

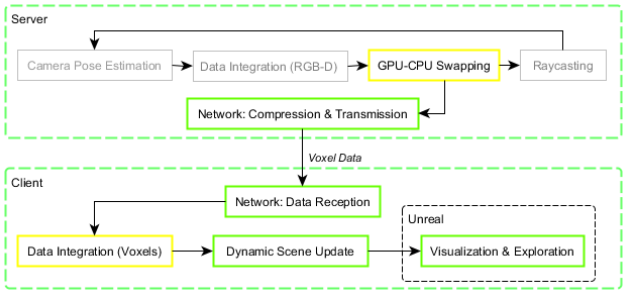

Figure 2: Overview of the immersive exploration framework. The server performs the scene reconstruction and sends the model incrementally to the client. The client maintains an up-to-date mesh representation and enables live exploration of the reconstructed scene. Unchanged InfiniTAM modules are gray, extended ones yellow and novel functionalities green.

Our proposed framework generates a dense 3D surface model from a real-world environment and incrementally streams this model over a (wireless) network to another workstation. From the very beginning of the capturing and reconstruction, the model can be explored in an immersive VR environment. All steps are real-time capable. Our framework's server performs scene reconstruction and uses internally Voxel Block Hashing [11] as data structure for scene representation. It holds significant advantage over a regular voxel grid, since information is not densely stored for every point in space, but only for regions which are close to a surface.

Voxel Block Hashing only stores a voxel block if at least one of its voxels is inside the truncation band of the Truncated Signed Distance Function (TSDF) [3]. The blocks are addressed using a spatial hash function, which maps a 3D voxel position from world space to a hash table entry. For data transmission, the server provides functionality to compress the voxel blocks and sends it to the client. The client listens for incoming volumetric data and assembles it; thereby, it has an exact copy of the server-side model. A mesh representation is favored by game engines such as UE4 over point clouds. Thus, a mesh is extracted from the volumetric representation and dynamically updated after integrating new voxel data. Next, the computed mesh is seamlessly passed to the visualization and exploration module that runs in a separate process. Within this module, users can interactively explore the 3D reconstructed scene.

For camera pose estimation and dense surface reconstruction, our proposed framework uses the original InfiniTAM [7] modules. For 3D visualization, we build upon the game engine Unreal 4 (UE4) [5]. To achieve streaming and live surface extraction, new modules were developed to 1) incrementally transmit the reconstructed model over a network and 2) perform on-the-fly mesh generation (and dynamic update) of the underlying model. Therefore, we furthermore extended the InfiniTAM modules *GPU-CPU Swapping* and *Data Integration* module to work with streamed data. To provide immersive exploration of the gradually expanding reconstruction, new features in UE4 were developed to support 1) communication with our novel framework to receive extracted meshes, 2) dynamic update of the scene representation, and 3) integration of the virtual reality input- and output devices.

In the following, we describe the methodology of the novel modules (green) and the extended InfiniTAM modules (yellow), both on server as on client side. For algorithmic details of the unchanged InfiniTAM modules *Camera Pose Estimation*, *Data Integration* and *Raycasting*, the reader is kindly referred to [7].

3.1 GPU - CPU Swapping

Generally, camera pose estimation and data integration is performed on the GPU to ensure real-time performance. To allow for reconstruction of large scenes, InfiniTam employs Moving Volume techniques so that only those voxel blocks that are currently processed are stored in the GPU memory. These voxel blocks lie within the current camera's view frustum, those that fall out the view frustum from frame t to $t+1$ are not required for processing and are transferred to CPU memory. Figure 3 illustrates both swapping in and out of voxel blocks.

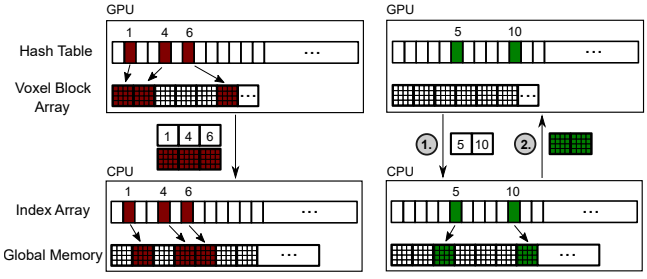

Figure 3: Swapping out to CPU (left), swapping in to GPU (right)

In InfiniTAM, the CPU memory that stores the complete model (global memory) contains exactly the same amount of elements as the voxel block hash table. To minimize memory footprint, we introduced an additional auxiliary array of indices with the same size as the hash table that is stored in the CPU memory. One element of this array forwards to an entry in the global memory. While this strategy slightly lowers the performance due to the additionally required lookups in the auxiliary array, the size of the global memory is greatly reduced since it does not store any empty voxel blocks. Furthermore, we stored the voxel blocks' world coordinate positions in the hash table as this is of major importance to be able to reconstruct the data on client side.

3.1.1 Network Transmission

To provide data transmission over the network, we developed a network module that allows to incrementally send the reconstruction data from the server over a (wireless) network to the client.

Server: Compression and Transmission Providing continuous streaming while keeping up with the model acquisition over time is of major importance to avoid data drop. With our proposed underlaying data structure, it is not critical if the model update on the client side is delayed by a couple of frames. Thus, we focused on lowering the required bandwidth while latency is secondary. This a clear advantage over streaming live images, which requires an interactive frame rate with low latency to avoid reconstruction failure on client side. By using Since Voxel Block Hashing, the mapped 3D space has been already efficiently compressed and individual voxel blocks can be independently addressed and processed, which is key for robust and efficient streaming. To further save bandwidth, we removed redundancy by solely streaming new or updated data. During the capturing process, only those voxel blocks that fall out of the current view frustum store new data and should be transmitted. Therefore, the server's network module waits for these voxel blocks which are identified by the GPU - CPU swapping module, gathers multiple voxel blocks in larger chunks to achieve a higher compression and groups TSDF, color and position to store common data types together. The three arrays are individually compressed using the lossless DEFLATE algorithm [4] and transmitted to the client. As network protocol, we employed TCP/IP to avoid data loss and to allow high throughput. In case of connection loss to the client, the network module waits until re-establishment and transmit all voxel blocks that have been reconstructed in the meantime.

Client: Data Reception and Integration The client listens for incoming data, decompress the individual arrays (TSDF, colors and positions) and assembles them to voxel blocks. With the given position of a voxel block, the client computes the corresponding hash value and checks for existence of another block at this position. If so, the TSDF and color values are overwritten with the new values. Otherwise, a new voxel block entry is allocated in the hash table and filled with the new data. The client stores the reconstructed scene as a lossless copy in the same format as the server. A hash table maintains voxel block positions and references to a voxel block array, which stores the actual voxel data. In contrast to the server, the representation is completely stored on the CPU memory and not (partly) on the GPU. Upon data integration, the *Scene Update* procedure is informed that new data arrived.

3.1.2 Dynamic Scene Update

To visualize and subsequently explore the 3D scene, a triangular mesh is computed from the underlying volumetric representation using Marching Cubes [9]. Therefore, we use InfiniTAM's meshing module that applies Marching Cubes for vertex position computation. We extended it by the computation of vertex normals and colors. Furthermore, we developed a new approach to allow dynamic mesh updates, that is described in the following.

Scene Subdivision To enable live exploration, the mesh needs to be updated upon model change. Inspired by Steinbruecker et al. [15], we propose a scene mesh representation that is composed of a number of smaller individual meshes, where each mesh covers a certain region of the volumetric scene. To ease the mesh computation, we store the volumetric scene only at one resolution. The scene is divided into a regular 3D grid of mesh blocks, where each

mesh block holds its own mesh and covers a region of n^3 voxel blocks. An example is shown in Figure 4. Real-time updates are achieved by recomputing an individual mesh, whenever new data for any of its underlying voxel blocks arrived. The triangulation of the voxel blocks is then performed using our extended Marching Cubes implementation.

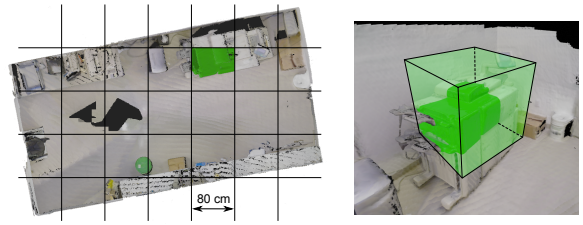

Figure 4: Scene partitioning into individual mesh blocks, where each block contains its own mesh.

The size of the mesh blocks is defined by the number of voxel blocks they should cover. For our setup, we found sufficient update rates with ten voxel blocks *vB* for each dimension per mesh. Note, that the metric size of a mesh block *mB* also depends on the chosen voxel resolution and number of voxels per voxel block. If one uses voxel blocks with 8^3 voxels at a resolution of $1cm$, one mesh block covers a volume of 80x80x80cm. To establish a correct mapping between the voxel blocks *vB* and their mesh block *mB*, each mesh block is identified by a non-negative and unique scalar index.

Algorithm 1: Compute mesh block ID

Data: P_{vB}, vB, mB
Result: $ID \in \mathbb{Z}$
$ID \leftarrow$ -1 ;
$P_{vB}(shifted) \leftarrow P_{vB} + \frac{mB}{2} \times vB$;
$ID_3 \leftarrow \frac{P_{vB}(shifted)}{vB}$;
if $(ID_3(x)$ ***AND*** $ID_3(y)$ ***AND*** $ID_3(z)) \in [0, mB]$ **then**
  $ID \leftarrow ID_3(x) \times mB^2 + ID_3(y) \times mB + ID_3(z)$;
end

This index *ID* can be computed from a given 3D voxel block position $P_{vB} \in \mathbb{Z}^3$, using the proposed Algorithm 1. Due to the division by the number of voxel blocks per mesh block, all voxel blocks within a mesh block produce the same mesh block ID. In order to allow this coordinate shifting, the maximum number of mesh blocks per dimension is predefined. Limiting *mB* to 1000 keeps the generated index below 1 billion (< 4 bytes), but still provides a sufficient large address space.

Algorithm 2: Compute mesh block position

Data: ID, vB, mB
Result: $P_{mB} \in \mathbb{Z}^3$
$P_{mB} \leftarrow (0/0/0)$;
if $ID >= 0$ **then**
  $ID_3(x) \leftarrow \frac{ID}{mB^2}$;
  $ID_3(y) \leftarrow \frac{ID - mB^2 \times ID_3(x)}{mB}$;
  $ID_3(z) \leftarrow ID - mB \times (ID_3(y) + mB \times ID_3(x))$;
  $P_{mB}(shifted) \leftarrow ID_3 \times vB - \frac{mB^2}{2} \times vB$;
  $P_{mB} \leftarrow P_{mB}(shifted) - \frac{mB^2}{2} \times vB$;
end

As given with Algorithm 2, one can reconstruct the world position of that mesh block $P_{mB} \in \mathbb{Z}^3$ using a given a mesh block index $ID \in \mathbb{Z}$. The client's network data reception module performs the index computation for every newly arrived voxel block to identify the mesh block to be updated. The computed index *ID* is checked for existence within the map of mesh block indices M_{ID}, which holds the indices of all mesh blocks to be updated. If the index does not exist, it is added to M_{ID}. The *Scene Update* module also accesses M_{ID}. It regularly reads and removes an index from the set. Given the index, the world position of the mesh block is reconstructed and the corresponding volume is triangulated.

Triangulation of a Mesh Block Upon model change, the triangulation of each mesh block is updated. For vertex position calculation, we employ InfiniTAM's Marching Cubes implementation. To furthermore compute the vertex color information, we apply an interpolation strategy in the same way as for positions and extract the vertex normals by performing a 3D gradient computation at the computed vertex positions. Instead of computing an array of triangle structures, our extended meshing procedure generates separate arrays for position, color, normal and vertex indices, where three consecutive indices define one triangle. This is the standard representation for rendering and is also required by UE4. Moreover, we employ a hash map to avoid duplicate vertices and thereby reduce memory footprint. Once the mesh is computed, it is passed to the *Visualization & Exploration* module that updates and maintains the mesh representation of the entire scene.

3.1.3 Visualization & Exploration

All meshes of the subdivided scene are stored and maintained by the rendering and exploration procedure, which runs as a separate process and uses UE4. We choose UE4 over Unity3D as it offers C/C++ programming, a powerful rendering engine and was released open source. Within UE4, the scene is represented as a set of procedural meshes, which can be added and modified at runtime.

Mesh Updating The UE4 process communicates with the *Scene Update* module using shared memory to enable an efficient data transfer. Upon update of a mesh block triangulation, the *Scene Update* module writes the data with a unique mesh block index to the shared memory. The UE4 process regularly checks for new data, reads updates and cleans the corresponding shared memory region. With the given mesh block index, UE4 can identify whether an existing procedural mesh requires an update or if a new one needs to be generated.

Immersive Exploration The 3D reconstruction can be explored from the very beginning of the scanning process while 3D scene viewing is decoupled from the view of the capturing device. To allow for immersive exploration, a HMD is integrated to support stereoscopic and egocentric viewing. We employed the *Oculus Rift Developer Kit 2* (DK2) that is nativly supported by UE4. For navigation, two different input devices were integrated to account for different level of immersion and means of navigation. The two-handed wireless gamepad *Xbox 360 Controller for Windows* was used to support navigation while the user is seated. Virtual locomotion is performed by moving the joystick or by using the arrow buttons, the walking direction is defined by the user's viewing direction, which is derived from the HMD's current orientation. To overcome the limitation of this abstract locomotion input and to provide proprioceptive feedback, we further integrated the *Cyberith Virtualizer* as omni-directional treadmill (ODT) to support natural walking. Thereby, the walking and viewing directions can be decoupled that leads to a more natural experience. To integrate the Virtualizer in UE4, we developed a UE4 plugin to retrieve body rotation, walking direction, speed and the attitude of the user's hip. This information is employed at run-time to update the virtual character's position and orientation. To enhance the visual appearance

of the 3D reconstruction during exploration, three artificial points lights are integrated in the virtual scene. Thereby, dynamic specular highlights are provided. Besides basic lighting calculation for those lights, no advanced rendering effects are applied to minimize computational burden and to maximize the rendering frame rate.

4 Performance Evaluation

To examine the technical properties of our proposed framework, we evaluated the performance of 3D reconstruction, network transmission and dynamic meshing. Our proposed framework has been implemented in C/C++ on Windows7 and integrates the following libraries: InfiniTAM 2 [7], OpenGL/GLUT (freeglut v3.0), OpenNI v2.2, CUDA v7.0, Windows Sockets v2, zLib v1.2, Unreal Engine v4.9, and Ultimate Shared Memory (usm). Due to the usage of Winsock and usm, our framework currently solely runs on Windows. The framework's performance was evaluated on a Windows7 notebook, featuring an Intel Core i7-4940MX processor (3.10 GHz), a GeForce GTX 980M graphics card as well as 16 GB of memory. Both client and server application are executed on the notebook, thus streaming is performed on the local machine which simulates a perfect network connection.

4.1 Testdata

For evaluation, three prerecorded camera streams are used to be able to reproduce the same scene multiple times. The data sets are OpenNI RBD-D sequences with depth as well as color resolution of 640×480 px. The employed data sets are shown in Figure 5 and are referred to as *Copy* room, *Lounge* and *Flat*.

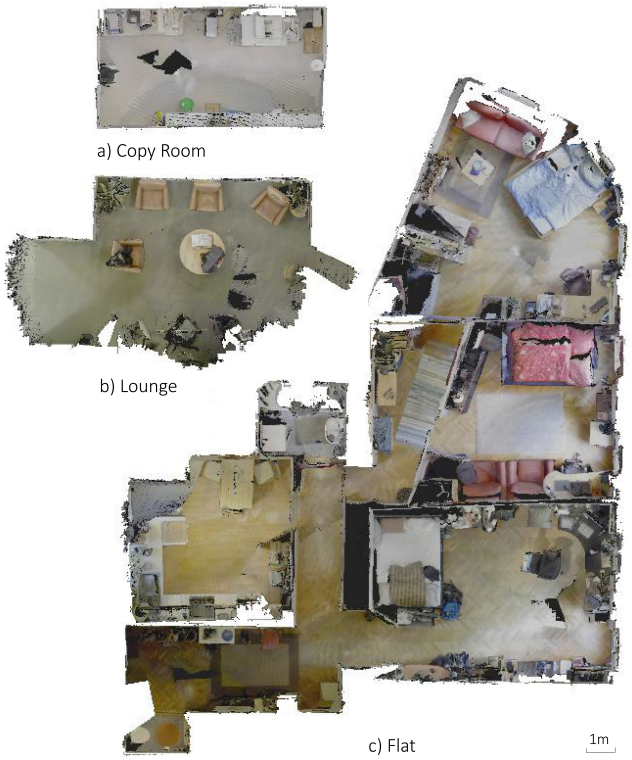

Figure 5: Top view on the reconstructions of the test data sets.

The single room tests sets Lounge and Copy were taken from [20], where the Copy room comprises 5490 frames (around 2:45 min) and Lounge 3000 frames (around 1:30 min). We furthermore captured a multi-room flat with a size of $93m^2$ that consists of 22175 frames (around 11:10 min) using an Asus Xtion Pro Live camera at $33Hz$. To stabilize the camera pose estimation and ensure a robust alignment of the large flat model, we employed precomputed camera trajectories [1] using the prerecorded camera frames of the flat. Besides this offline procedure, all other framework's computations were performed in real-time during the evaluation. The framework's processing pipeline was executed eight times for each test data set, with varying the voxel size $S_{vox} = 0.5cm$, respectively $1.0cm$, and the width of TSDF truncation band $\lambda = 2.0cm$, respectively $4.0cm$.

4.2 Experimental Results

In the following, we report our performance results. Over all test cases, the rendering performance of UE4 achieved 200 frames per second, a drop of the frame rate during a mesh update was not noticeable. Furthermore, we focus on analyzing memory footprint during 3D reconstruction, network transmission and dynamic meshing.

3D Reconstruction To be real-time capable, 3D scene reconstruction must be processed at least at the camera's capturing rate, resulting in a maximum processing time of $30ms$ for each frame. On our system, the reconstruction process takes on average $5ms$ for pose estimation and data integration, more details on processing speed are given in [7]. To examine our novel functionalities, we analyzed the required memory that is needed both on server and client side to store a reconstructed scene with the employed volumetric data format. Using InfiniTAMs default hash table size with a maximum of 1179648 addressable entries, the required memory was around 100 MB for the single room environments (Copyroom 97 MB, Lounge 104 MB) and 811 MB for the flat. These footprints were found for a colored reconstruction with a voxel resolution of $S_{vox} = 1cm$ and a truncation band width of $\lambda = 2cm$. Memory footprint increases in average by 30%-40% when setting the truncation band width to $4cm$. This is due to the fact that more voxel blocks fall into the truncation band and have to be processed. The major impact on memory footprint has the voxel size. A resolution of $S_{vox} = 0.5cm$ increases the hash table occupancy and the required memory by a factor of around six. The Flat data set could not be completely processed, thus we increased the hash table size by a factor of four. With these setting, we could completely reconstruct the Flat scene, up to 1.85 million voxel blocks were allocated which required for a colored reconstruction 5 GB ($\lambda = 2cm$), respectively 7 GB ($\lambda = 4cm$).

Compression & Network Transmission Each transmitted voxel contains TSDF, color and weight, the latter can be excluded to save bandwidth requirements. This lead to 2, respectively 5 bytes per uncolored/colored voxel instead of 6, respectively 8 bytes. We tested solely for the actual network payload and did not consider perform tests that includes any overhead created by TCP/IP. The transmission data rates, both average and maximum rates in MBit/s, are listed in Table 1.

Parameters		Colored Model		Uncolored Model	
S_{vox}	λ	Avg. MBit/s	Max. MBit/s	Avg. MBit/s	Max. MBit/s
1cm	2cm	3.1	16.5	1.7	11.7
1cm	4cm	4.3	24.2	2.5	14.8
0.5cm	2cm	23.0	113.8	12.4	68.8
0.5cm	4cm	30.7	175.0	18.0	110.6

Table 1: Average and maximum data rates of all three scenes.

A colored reconstruction with $S_{vox} = 1cm$ and $\lambda = 4cm$ led to an average data rate of 4.3 MBit/s, however peaks occurred up to 24.2 MBit/s. This is caused by rapid scanning movements, such as quick turns which result in an increased amount of voxel blocks that fall

out of the frustum and need to be transmitted. The data rate can be lowered by decreasing the sensor's depth range; in our tests we set the depth range to $3m$ to reduce the probability of reconstruction artifacts since the depth error quadratically increases with the depth range. In general, the data rate for colored reconstructions has been found to be increased by a factor between $1.5 - 2.0$ than for uncolored reconstructions, while a finer voxel resolution increases the data rate by a factor of seven, and an enlarged truncation band width increase the data rate up to 40%.

Dynamic Meshing We tested the *Scene Update* procedure in terms of processing speed and properties of the computed meshes. In average, the *Scene Update* module requires $65ms$ for triangulating the data within one mesh block. To achieve higher update rates for triangulation, we ran the meshing with multiple threads. Using four threads during the reconstruction of the Flat led to 40,000 updates of the entire mesh representation, compared to 10,000 when running the procedure single-threaded. As a consequence, the mesh representation grows more smoothly and changes in the volumetric model are reflected earlier.

Parameters		Mesh Representation			
Scene	S_{vox}	# Meshes	Memory	Vertices	Triangles
Copy	1cm	128	47 MB	0.98 M	$1.81*10^6$
Copy	0.5cm	582	212 MB	4.42M	8.25M
Lounge	1cm	142	45 MB	0.95M	1.71M
Lounge	0.5cm	578	216 MB	4.52M	8.32M
Flat	1cm	724	270 MB	5.71M	10.29M
Flat	0.5cm	3335	1245 MB	26.07M	47.97M

Table 2: Average and maximum data rates of all three scenes.

In Table 2, an excerpt of the computed mesh properties properties is listed, as we found no significant differences between $\lambda = 2cm$ and $4cm$. The number of individual meshes corresponds to the amount of employed mesh blocks, furthermore memory footprint and total amount of vertices and triangles are stated. The major effect on the total number of meshes has voxel size. In average, the number of meshes is four to five times higher when using the smaller voxel size of 0.5 cm.

5 Conclusion & Outlook

In this paper, we introduced a novel framework for large-scale 3D scene reconstruction, streaming and immersive virtual exploration by two remote entities. To enable this, it partly builds upon readily available components [7, 5], extends existing modules and integrates newly developed approaches. We analyzed the performance of our proposed framework by focusing on 3D reconstruction, network transmission and dynamic meshing. The performance evaluation revealed the framework's capabilities to perform RGB-D data capturing, dense 3D reconstruction, streaming and dynamic scene updating in real time for indoor environments up to a size of $100m^2$, using either a state-of-the-art mobile computer or a workstation. Our presented work provides a foundation for enabling immersive exploration of remotely captured and incrementally reconstructed dense 3D scenes. We plan to release both framework and data for the community to foster research on employing virtual reality technology for interaction with streamed dense 3D environments.

There are some current limitation we aim to overcome in future. We have found significant drift in the reconstruction of large-scale models in areas that are revisited, such as connecting hallways. For camera pose estimation and data integration, we incorporated the InfiniTAM modules that currently do not support loop-closures. Integrating algorithms for loop-closure detection and subsequent dynamic mesh updates are important and thus, a major goal for the future. Furthermore, the evaluation of capturing, reconstruction and streaming of multi-storey buildings with our framework is subject to future work. Therefore, collision detection within the reconstructed model must be enhanced which can be achieved by employing the generated procedural meshes as colliders.

References

[1] S. Choi, Q.-Y. Zhou, and V. Koltun. Robust Reconstruction of Indoor Scenes. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5556–5565, 2015.

[2] M. Coatsworth, J. Tran, and A. Ferworn. A hybrid lossless and lossy compression scheme for streaming RGB-D data in real time. In *12th IEEE International Symposium on Safety, Security and Rescue Robotics, SSRR 2014 - Symposium Proceedings*, 2015.

[3] B. Curless and M. Levoy. A Volumetric Method for Building Complex Models from Range Images Volumetric integration. In *SIGGRAPH*, pages 303–312, 1996.

[4] P. Deutsch. DEFLATE Compressed Data Format Specification version 1.3. Technical report, Network Working Group, 1996.

[5] Epic-Games. Unreal Engine. [Software] (v4.9), http://www.unrealengine.com, Last accessed: February 11 2016.

[6] T. Golla and R. Klein. Real-time point cloud compression. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 5087–5092. IEEE, 2015.

[7] O. Kähler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. Torr, and D. Murray. Very High Frame Rate Volumetric Integration of Depth Images on Mobile Devices. *IEEE Transactions on Visualization and Computer Graphics*, 21(11):1241–1250, 2015.

[8] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. *Eurographics Symposium on Geometry Processing*, 32(3):61–70, 2006.

[9] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 21(4):163–169, 1987.

[10] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. *10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, oct 2011.

[11] M. Niessner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3D Reconstruction at Scale Using Voxel Hashing. *ACM Trans. Graph.*, 32(6):169:1—-169:11, 2013.

[12] H. Roth and V. Marsette. Moving Volume KinectFusion. *Proceedings of the British Machine Vision Conference*, pages 112.1—-112.11, 2012.

[13] R. A. Ruddle and S. Lessels. The benefits of using a walking interface to navigate virtual environments. *ACM Transactions on Computer-Human Interaction*, 16(1):1–18, 2009.

[14] F. Steinbrucker, C. Kerl, J. Sturm, and D. Cremers. Large-scale multi-resolution surface reconstruction from RGB-D sequences. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3264–3271, 2013.

[15] F. Steinbrücker, J. Sturm, and D. Cremers. Volumetric 3D mapping in real-time on a CPU. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2021–2028, 2014.

[16] Visual Computing Lab of ISTI - CNR. MeshLab. [Software] (V1.3.3) http://meshlab.sourceforge.net/, Last accessed: October 6 2015.

[17] A. Wendel, M. Maurer, G. Graber, T. Pock, and H. Bischof. Dense reconstruction on-the-fly. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1450–1457, 2012.

[18] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, J. Mcdonald, and J. J. Leonard. Kintinuous : Spatially Extended KinectFusion. Technical report, MIT, Boston, MA, USA, 2012.

[19] T. Whelan, L. Ma, E. Bondarev, P. H. N. De With, and J. McDonald. Incremental and batch planar simplification of dense point cloud maps. *Robotics and Autonomous Systems*, 69(1):3–14, 2015.

[20] Q.-Y. Zhou and V. Koltun. Dense scene reconstruction with points of interest. *ACM Transactions on Graphics*, 32(4):112:1—-112:8, 2013.